Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Jere Nevalainen

A First Person Immersive Animation Tool

Master's Thesis
Espoo, May 11, 2015

Supervisor: Assistant Professor Perttu Hämäläinen

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Jere Nevalainen

Title:
A First Person Immersive Animation Tool

Date: May 11, 2015 **Pages:** viii + 60

Major: Media Technology **Code:** T-111

Supervisor: Assistant Professor Perttu Hämäläinen

Computer-generated animation has an important role in both video game and film industry. 3D computer animation is generally done with 2D devices, such as the computer mouse, that are not optimal for that kind of use. To use them efficiently in 3D, long training periods are needed. This makes them especially inefficient in novice hands. Additionally, 2D display devices, such as the computer monitor, are unable to give the user depth perception. Therefore the user has to rely on perspective projection.

The goal of this thesis was to create an animation tool, which provides the user a 3D view of the objects of interest and the user can manipulate the objects with his own hands in virtual space. This should be quite natural for users, because humans have lived their whole lives in a 3D world. The tool was developed and tested in collaboration with professional game industry animators.

The tool uses an Oculus Rift DK2 device to give the user a 3D view with 6 DOF head tracking. The hand tracking is done with a Leap Motion controller, which is mounted in front of the Oculus Rift device. The software was programmed using the Unity game engine.

According to the results there are uses for this sort of a tool especially in the first rough posing phases. The hand tracking quality still has something to improve, but even at the current level it can increase productivity in certain parts of the workflow.

Keywords: animation, virtual reality, oculus rift, leap motion, hand tracking, hand gestures

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Jere Nevalainen		
Työn nimi:	Ensimmäisen persoonan immersiivinen animaatiotyökalu		
Päiväys:	11. toukokuuta 2015	**Sivumäärä:**	viii + 60
Pääaine:	Mediatekniikka	**Koodi:**	T-111
Valvoja:	Apulaisprofessori Perttu Hämäläinen		

Tietokoneella luotu animaatio on tärkeässä osassa videopeli- ja elokuvateollisuudessa. Kolmiulotteisia tietokoneanimaatioita tehdään yleisesti kaksiulotteisilla laitteilla, kuten hiirellä, jotka eivät ole optimaalisia tähän käyttöön. Niiden käyttäminen tehokkaasti vaatii pitkää harjoittelua, joten ne ovat epätehokkaita etenkin vasta-alkajien käsissä. Kaksiulotteiset näyttölaitteet eivät myöskään pysty antamaan syvyysvaikutelmaa, joten käyttäjät joutuvat tyytymään perspektiiviprojektioon.

Tämän diplomityön tarkoituksena oli luoda animaatiotyökalu, jonka avulla animaattori näkee hahmon kolmiulotteisena ja voi muokata sen asentoa omin käsin perinteisen nukkeanimaation tapaan. Tämän pitäisi olla hyvin luontevaa, koska ihmiset ovat oppineet elämään kolmiulotteisessa maailmassa. Työkalu kehitettiin yhteistyössä peliteollisuudessa työskentelevien animaattoreiden kanssa.

Työkalu käyttää Oculus Rift DK2 -laitetta antaakseen käyttäjälle kolmiulotteisen näkymän ja kuuden vapausasteen liikkeentunnistuksen pään asennoille. Käsien liikettä seurataan Leap Motion -ohjaimella. Työkalun ohjelmointi tehtiin Unity-pelimoottoria käyttäen.

Tuloksien perusteella tämänkaltaisella työkalulla olisi käyttöä etenkin karkeassa alkuvaiheen sommittelussa. Käsienseurantalaitteen tarkkuudessa on vielä parantamisen varaa, mutta nykytasollakin pystyy tehostamaan tiettyjä kohtia työnkulusta.

Asiasanat:	animaatio, virtuaalitodellisuus, oculus rift, leap motion, käsien seuranta, käsieleet
Kieli:	Englanti

Acknowledgements

I would like to thank my thesis supervisor, professor Perttu Hämäläinen for the original idea for this thesis and all the following input. The whole project was very interesting and I learned a lot in the process.

I would also like to thank all the testers who gave their input during the prototyping phase and the testers who participated in the final user test. The future work ideas I received were very valuable.

I thank the computer science guild Tietokilta ry for providing me seven years of networking and joy during my studies. The years in the board and various committees were an excellent balance to study and work life.

Finally I want to thank my family, friends and especially Gaja for all the support during this whole project. It was quite a ride.

Espoo, May 11, 2015

Jere Nevalainen

Abbreviations and Acronyms

DK2	Development Kit 2
DLL	Dynamic Link Library
DOF	Degrees of Freedom
FBX	Filmbox (file format)
FOV	Field of View
HCI	Human-Computer Interaction
HMD	Head Mounted Display
Lerp	Linear Interpolation
NCF IK	Non-Iterative, Closed-Form, Inverse Kinematic Chain Solver
SDK	Software Development Kit
Slerp	Spherical Linear Interpolation
SUS	System Usability Scale
VR	Virtual Reality

Contents

Chapter 1

Introduction

Three dimensional (3D) computer animation has become a staple of movie and video game entertainment. Computer animation's roots are deep, and for a long time the animations have been created with tools that can work only in two dimensions (2D): the computer mouse, keyboard and the traditional 2D computer display. These are not optimal when working with 3D worlds. There are alternative ways to create high quality animation data such as motion capture technologies. The problem with motion capture technologies are the generally large space requirements, the need of actors and long calibration procedures. Such things are not always possible, so the mouse and keyboard combination is a solid option.

When it comes to working on a desktop, various different vision based and mechanical devices have been invented that are supposed to be more suitable for 3D work. These offer more degrees of freedom (DOF) than the popular computer mouse, which is confined to working in two dimensions. Plenty of the devices have stayed on a proof-of-concept level and have never found their way into professional usage.

Lately virtual reality (VR) head mounted display (HMD) devices have gained a boost in popularity with the introduction of consumer grade devices such as the Oculus Rift Development Kits 1 and 2 and the upcoming SteamVR by Valve Corporation and HTC Corporation. These devices bring 3D virtual worlds with head tracking to homes and offices at a very affordable price.

Hand and tool tracking has been researched for decades. The resulting devices have had little presence in the input device market compared to the dominating input devices: the computer keyboard and mouse. Lately a hand

tracking device called the Leap Motion controller was introduced and it offers 6 degrees of freedom tracking for hands and tools. Research shows that 6 DOF tasks such as positioning and rotating objects in virtual space should be done with 6 DOF tools, so using the device for animation purposes is an attractive idea.

The goal of this thesis was to build an animation tool that uses the latest easily available virtual reality hardware and to see how well current virtual reality technologies fit into animation production. The combination of 3D display, head tracking and hand tracking hopefully offers the user a very natural way to interact with a poseable 3D character. The tool should not require a lot of space around the user and setting up the system should not require a lot of calibration every time the system is brought into use. The tool is especially good for novices, since most animation software has quite steep learning curves because the interaction with mouse and keyboard is not natural when working with 3D objects. When using these existing tools, the user has to learn how his or her 2D actions map to the 3D world. Bringing the user's hands into the 3D world and giving the user a 3D presentation of the world hopefully make the learning process much faster, since the user has already learned how to interact with 3D objects in real life with his or her hands.

In Chapter 2 we describe the goals of this thesis more thoroughly. In Chapter 3 we go through different interaction techniques in 3D and how different input and display devices have been used in 3D animation. In Chapter 4 we describe the hardware and software used in this thesis project. In Chapter 5 we describe how the required features were implemented and how the hardware functions were used in the software. In Chapter 6 we evaluate how suitable the created tool is for animation production. Lastly, in Chapter 7 we discuss the conclusions and think of possible future work.

Chapter 2

Goals and requirements

In this chapter we go through the goals and requirements for our project.

2.1 Goals

The goal was to create a first person immersive animation tool using current easily available hardware that hopefully feels very natural for the users to use. We found out how well it fared in animation creation and how animation professionals felt about using such devices in their work.

2 DOF mouse and keyboard have been dominating the industry for a long time, although they are not optimal for 6 DOF 3D manipulation (see chapter 3). Using 6 DOF devices for 6 DOF tasks presumably decreases task completion times, but not necessarily accuracy, because the computer mouse and keyboard have been found to be very precise devices [19]. We asked computer animation professionals to test the system and to provide feedback on how well they could see this sort of a system being used in the industry and for what sort of tasks.

2.2 Requirements

The following requirements were defined at the start of this thesis for the character posing tasks:

- The user needs to be able to freely move the animated character above a plane that is designated as the floor.

- The character itself should be moved by grabbing the root body part, which is for humanoid character usually the hip region.
- Other body parts should be rotatable and twistable.
- Arms start moving from the shoulders, legs start moving from the hips, head from the chest and spine from the hips.
- Fingers and toes are not movable, since there is no scaling in this version of this project. Fingers and toes are too small to be grabbed reliably.

For timeline control and keyframing, following requirements were defined:

- The user creates keyframes and the tool will interpolate the frames between them.
- The user can also modify and delete these keyframes at will.
- For body part rotations spherical linear interpolation (Slerp) will be used and for the root positions linear interpolation (Lerp) is used.
- The user must be able to manually scroll through the animation by scrubbing the timeline. This means the user must be able to jump into any point of the animation and insert new keyframes or edit existing keyframes.
- The length of the whole animation, therefore also the length of the timeline, must be adjustable by the user.

Additionally, the viewpoint must be adjustable. If the default viewpoint is set in front of the animated character, the user must be able to look at the character from any angle and distance.

Originally it was planned to be able to be able to export animation data in Filmbox (FBX) format, but as the project went on this idea was scrapped due to time and software constraints. Unity 4.6 Free edition has limitations on how it can handle external Dynamic Link Libraries (DLL). Unity 4 Free can handle managed DLL libraries, which means it only supports .NET assemblies. Exporting in FBX requires use of the FBX Software Development Kit (SDK) DLL, which is an unmanaged C++ library and therefore requires a Unity 4 Pro license. At a late part of this thesis project Unity 5 was released and it can use the FBX DLL for free, but it was too late at that point.

Chapter 3

Interaction in computer animation

The field of computer animation is filled with different techniques and devices for creating realistic and imaginative animations. Some favor ease of use and speed over quality, while some systems use a lot of resources, because sometimes only the finest quality is acceptable.

The first section 3.1 of this chapter concentrates on what requirements interaction in 3D and animation has and how different interaction techniques manage to meet them. The next section 3.2 looks at different input devices and talks about the pros and cons of each one when interacting with 3D objects or the user interface. Lastly, interacting with computers without any feedback is quite difficult. There is a wide variety of different output devices and some of them are more suitable for 3D work than others. These are discussed in the section 3.3.

3.1 Human-computer interaction techniques in 3D

When building a user interface, no matter if it is for 2D or 3D work, the interface should be intuitive and easy enough to use so the user can concentrate on learning the task domain instead of fighting the interface syntax or rules. [22] Human capabilities should be taken into consideration when designing how the user will interact with the virtual presentations. There is plenty of research into how human sensorimotor system works and how we

handle 3D tasks. All this data should be taken into consideration so the user interface can be not only intuitive but also natural for the target demography. The more natural and transparent the experience is, the more engaged and productive the users are. [14]

In real life 3D objects can be moved around, rotated and depending on the material the objects can also be reshaped. Being able to do these things intuitively in a virtual space increases immersion and makes the users feel like they are really there. The user interface fades away and the users are left alone with their tasks. Virtual spaces also add things that can not be done in real life, such as creating objects from nothing, destroying them without a trace and scaling them freely.

Just manipulating objects is not enough when working with 3D objects and animations. If the application is interactive, the users generally have to be able to change their viewpoint at will. They need to be able to move around in the virtual space and circle around points of interest, so they can for example see the animation they have been working on play back from different angles. Working on small details, such as a character's finger movements, is difficult or in some cases impossible without zooming features.

On top of manipulating objects and moving around in the virtual space, the user usually needs to be able to control the application he or she is working with. This can be as simple as shutting down the application or in some cases complex, such as when the user needs to switch between tools or modify parameters that affect the 3D scenes he or she is working on. Even if some interaction techniques or devices are exceptionally good when manipulating virtual 3D objects, they might be extremely cumbersome or inefficient when controlling the application itself. This might introduce issues such as increased amount of interaction devices and the need for the user to switch between them and adjust to each of them.

Important application control issue in computer animation is timeline scrubbing. The users need to be able to move forward and backward on a timeline so they can observe how any changes they have made affect the animation. The users also need to have some sort of control on what sort of poses the characters have at certain times. One way to do these are keyframes. The user poses a character and then tells the application to save the current situation as a keyframe. After the user has set different keyframes at different points on the timeline, the software can interpolate poses for the frames between the keyframes.

Different interaction techniques have been developed to aid human and

computer co-operation. Each of them have areas where they are better than others. We will have a look how common techniques can handle the tasks that arise when working with 3D objects and animation software. Human-computer interaction methods are commonly split into different styles: command language, menu selection and direct interaction [22] with sometimes form fill-in and natural language as fourth and fifth category [29]. These methods can be used alone or mixed together.

3.1.1 Command based interaction

In command based interaction the user inputs text commands in a text field or a command line. The commands may have extra variables appended to the end that change what the command does. The advantages and disadvantages of a command based interface are [29]:

- Advantages
  - Flexibility
  - Appealing to advanced users
  - Support for user initiative
  - Allows creation of user-defined macros
- Disadvantages
  - Poor error handling
  - Requires substantial training and memorization

Command based interaction is not very suitable when dealing with objects in 3D space. The user needs a high level of expertise to know how the object reacts to commands and variables that he or she inputs. Example of a command that moves an object to a position in a 3D space could be *translate 50 50 0* where *translate* is the command to move the object to a position that is given by the trailing numbers, as XYZ-coordinates in this example. The user has to know how much one coordinate unit moves the object in the 3D space in question. Command based interaction is very precise and fast if the user happens to know exact coordinates beforehand, but even in that case the used interaction technique is probably form fill-in described in section 3.1.3.

Since actions have their own commands and some commands might even combine multiple actions into one command, the user has to learn all relevant commands by heart or refer to a manual every time infrequent commands are needed. This makes this interaction quite slow to use for rookies, but seasoned veterans can use the full flexibility and power of this technique.

3.1.2 Menu selection

Menu selection systems provide the user with lists of actions. The user can pick the best action for the current situation. As long as the list has sensible terminology and the available actions are comprehensive enough, the user can get through their task with minimal learning, especially when compared to command based systems. Shneiderman [29] lists the menu selection advantages and disadvantages as:

- Advantages
  - Short learning times
  - Reduces keystrokes
  - Helps structure decision making
  - Permits use of dialog-management tools
  - Easy support of error handling
- Disadvantages
  - Actions might get lost in case of several menus
  - May be slower for advanced users than for example command language
  - Consumes screen space
  - Requires a fast enough display rate

Using menu selection to manipulate 3D objects has the same problems as command based interaction. Menu selection is essentially the same thing, the command are just laid down for the user to use with the mouse or keyboard shortcuts. Menus are better suited for selecting tools or modes that are then used with direct manipulation. More on direct manipulation comes later in section 3.1.5.

3.1.3 Form fill-in

When using form fill-in techniques, the users literally fill forms the user interface asks them to. Form fill-in usually accompanies other interaction techniques, since it is geared towards data entry rather than telling the computer what actions to do. For example some menu selected action can give the user a dialog box with forms so the user can give more specifications what the action is supposed to do. Form fill-in advantages and disadvantages are [29]:

- Advantages
  - Simplifies data entry
  - Modest training required
  - Can give assistance to the user
  - Permits use of form-management tools
- Disadvantages
  - Consumes screen space

Form fill-in is very common in modern 3D capable applications. For example the Unity editor interface shows a frame window for the currently selected object and the window includes information about the object. Some of the information can not be changed directly, but the form contains fields for variables such as position, rotation and scale. The user can directly input the values he wants without dragging the object around on the screen. This is faster and more accurate than manipulating the object with a mouse for example, but the values have to be known beforehand. Otherwise finding the correct position or rotation probably takes multiple tries and a lot longer than with direct interaction through the object.

3.1.4 Natural language

Natural language is completely different from the techniques mentioned before. In natural language systems the user simply writes or orates natural language sentences and the computer is supposed to understand what to do. For example the user could say *"Make a kneeling pose with both hands up in the air"*. Without any rules or syntax the amount of possible interpretations for the commands is immense and there has been little success so far. [29] Natural language will not be covered more as it is not particularly relevant yet.

3.1.5 Direct manipulation

Direct manipulation systems are more visual and rapid than the systems discussed so far. In direct manipulation systems the objects of interest are kept visible at all times. Secondly, any complex syntax is replaced by physical actions or well labeled buttons. Finally, operations consist of many rapid and easily reversible actions and their effects can be seen on the objects in real time. [15] Direct manipulation advantages and disadvantages are [29]:

- Advantages
  - Visually presents task concepts
  - Easy learning
  - Easy retention
  - Allows errors to be avoided
  - Encourages exploration
  - Affords high subjective satisfaction
- Disadvantages
  - May be hard to program
  - May require graphics display and pointing devices

The visualness of direct manipulation is obviously a very attractive concept when designing an user interface for 3D interaction. The rapid and iterative approach is also effective when posing characters for an animation. If the first pose is not quite what the user wanted, he or she can easily fine-tune the pose or reverse his or her actions easily. Application control task such as moving on a timeline can be done directly by clicking on a point on the timeline and the time jumps to that position, or the user could drag a marker that shows the current position around.

When using physical input devices for animation and posing purposes, direct manipulation is the most desirable of the described interaction techniques. Major reason for this is that animation is always visual and with direct manipulation the users can always see how their actions affect the character on the screen. Especially when using head mounted display (HMD) devices that place the user in the virtual world, directly manipulating the objects the user actually sees in 3D is as natural it can get.

3.2 Input devices

Ultimately all input devices are devices that encode motion, sound or other wave data into a signal that can be read by a computer. Since computers have so many different uses, fundamentally different input devices have been developed over time. Some are more generalistic that have many purposes, while some have been designed to do specialized tasks.

In this section we cover current and past input devices that have been used when dealing with objects in 3D space. User interface usage is also touched, if the device is especially suitable for that. Some of the devices, such as the mouse and keyboard, are common in general everyday computer usage, while some of them are more experimental or especially geared towards 3D work.

3.2.1 Keyboard

In 3D interaction, keyboard offers more experienced users large degree of control and accuracy at the price of sacrificing ease of use [19]. Since the keyboard is mostly used to input exact values in fields the interface provides, the user has to already know the values or at least be able to roughly guess the correct values and fine tune them later. If the user is unfamiliar how the interface maps the values to the 3D world, adjusting objects such as character limbs may require multiple tries to get results even close to acceptable.

The keyboard can also be used to manipulate objects directly without inputting any values. Keyboard's keys can be mapped into operations that manipulate objects in the virtual world. For example, six keys can be mapped as positive and negative motions on X, Y and Z axes and similarly six other keys for rotations. This might sound a lot to remember for something that is very natural for people to do with physical objects in real life.

In real world we can move and rotate objects relative to their current position by picking them up and simply moving them. With keyboard the movement has to be done by pressing a button that changes the position or rotation incrementally. All interaction with a keyboard that is not about inputting values into fields is always incremental. This takes away from the naturalness.

Keyboards excel in user familiarity because just as the computer mouse, most users learn to use them as soon as they start using the computer. Since nearly every computer system has a keyboard, most computer software

supports them. In 3D software the keyboard's part is more on the supportive side. The keyboard is used to input shortcuts to change tools or usage modes, so an input device more suitable to handle 3D interaction can be used more efficiently.

3.2.2 Mouse

The computer mouse, just like the keyboard, offers a high level of precision and control. These days, the mouse should be familiar to anyone who is interested at doing computer animation and all the major animation software have user interfaces that are generally used with a mouse. Even though the mouse has pervaded the animation scene so widely, we can show some reasons why it is not necessarily the best option for 3D posing and animation.

Since the mouse is so prevalent in current computer setups, most of the successful 3D modeling software have user interfaces designed for the mouse. The user interfaces usually have menus and tool buttons laid on the screen around a window that contains a scene where the virtual objects lie. At least a set of most important tools also have keyboard shortcuts, but the interfaces are generally best used with the mouse and can not be used with some interaction devices if they can not control the mouse cursor and send signals that imply a mouse click.

Selecting objects with a mouse in 3D environments is generally done by ray casting. A ray is cast from the viewpoint through a pixel the mouse is pointing at. The object the ray intersects with first is selected. This has been found to be an efficient and easy to learn way to select the objects. [31] How these selected objects can be manipulated is talked about in the section 3.2.2.1.

The generic computer mouse has 2 DOF. The mouse can be slid on a table in any direction parallel to the table surface, but the movement will still be mapped to two coordinate axes. These axes can be for example X and Y. If translating an object in 3D space in a way that all three of its Cartesian coordinates change, the mouse can not perform the translation in just one action [18]. The first action translates the object's two coordinates (let them be X and Y) and on the second action translates the final coordinate. How the affected coordinates are chosen depends on the software. Common way is to use transform gizmos that display the axes on screen and one or two of the axes can be chosen for modifying. More on transform gizmos can be found in section 3.2.2.1.

There have been more exotic computer mice with more than 2 DOF. Two of these are the Cubic Mouse by Fröhlich et al. [11] and the Rockin' Mouse by Balakrishnan et al. [1] These are not shaped like the computer mouse we know today and they did not become a staple computer peripheral. However, testers of both mice had faster results when working with 3D objects and preferred the more exotic mice over the traditional mouse.

One advantage of a computer mouse is that using one leaves the non-dominant hand free for other tasks and devices. This way the user can enhance his productivity with for example a keyboard that is used to input keyboard shortcuts. These shortcuts can change tools the user controls with the mouse, change between different viewpoints or switch between programs altogether. This eliminates the time needed when using the mouse to go through menus or click tool buttons in the interface to change interaction modes or tools.

3.2.2.1 Axis Selection in Mouse Based Interaction

Computer mice generally have two degrees of freedom, but translation, rotation and scaling each have three adjustable variables. Because of this there has to be a way to select which variables the mouse movements modify. There have been different solutions for this ranging from keyboard shortcuts that toggle between modes, to on-screen tools that are used with the mouse itself.

Most effective way to apply transformations to objects on the scene with a mouse is to use manipulators that are on the scene with the objects. When the user manipulates the objects, the manipulators move along with them and the user's attention stays on the object. They also separate the required 3D operations into simpler to use 1D or 2D operations. Moreover, well implemented manipulators give graphical hints how their actions affect the objects. [20]

In modern 3D capable applications such as Unity, Maya or 3ds Max these transformation widgets are usually called *transform gizmos*. A picture of the transform gizmos in Unity can be found in figure 3.1.

The gizmos for translate and scale operations are heavily based on skitters and jacks principle by Bier [3]. He describes skitters as cursors that visually show positive vectors for all three axes of the object. The skitter can be on the surface or inside of the object. Jacks are the same thing but with negative vectors added. In figure 3.1 the green, red and blue arrows or cubes for translate and scale gizmos show these positive axes. If the user clicks

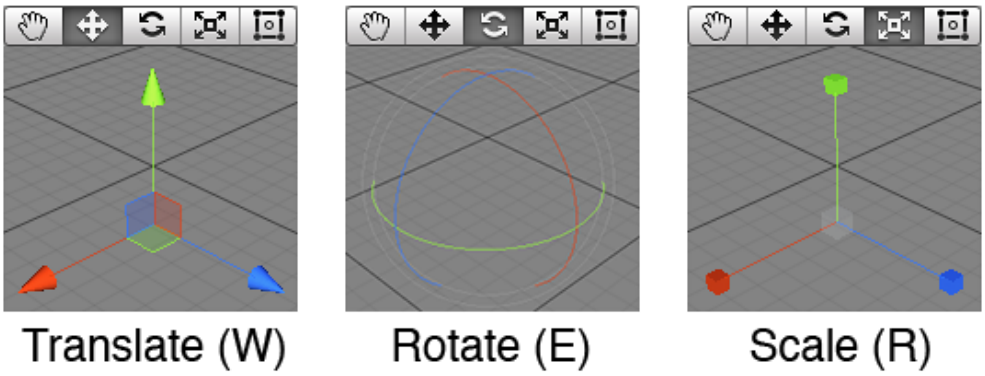


Figure 3.1: Unity transform gizmos and their keyboard shortcuts as seen in Unity documentation[34]

any of them and drags the mouse, the object will be translated or scaled in that direction. If the user clicks on the plane perpendicular to two axes, the mouse dragging applies to both axes at the same time. This means the user can at any time only manipulate two axes at the same time. To affect all three axes the user needs at minimum two operations.

The rotation gizmo is based on Bell's Trackball [2], the Virtual Trackball by Chen et al. [8] and the Arcball by Shoemake [30]. Each of these create a visual ball inside or around the object that is to be rotated. The user can rotate these virtual balls with the mouse cursor as they would rotate a physical trackball mouse. This rotation maps to the object and it moves accordingly. In the modern rotation gizmo, if the user clicks on any of the colored lines around the ball and drags the mouse, the object will rotate along the selected axis. When dragging elsewhere on the ball, which axes are affected depends on the viewpoint and which direction the mouse is moved to, but in the end the rotations apply to maximum two of the axes simultaneously.

3.2.3 Physical input devices with varying degrees of freedom

Since mouse and keyboard are clearly unideal for 3D work, it is not surprising that multiple solutions have been devised to aid users in their work. They range from simple knobs and sliders to sophisticated systems that track human hand position and orientation in full 6 DOF.

Knob boxes have been used with 3D tools such as CAD software. They

are boxes or planks that have knobs for each movement and rotation axis. Although these devices have high precision, the interaction style is far from natural for human beings, who are used to directly manipulating objects with hands. When interacting with 3D objects, we should be able to take advantage of the skills we have learned when growing up in a 3D world. [13] This means the devices should somehow take hand movements into account in an integral way, not separable. Tasks such as positioning an object in 3D space is done fastest with integral tools [18] and devices that take advantage of the user's visual and proprioceptive abilities have best hand-eye coordination results [21].

There are different ways to track how the user uses his or her hand. Common ways are mechanical, electromagnetic, optical, acoustic and intertial tracking [13]. Mechanically tracked devices have a handle the user grabs and the handle is connected to an arm, that has sensors in its joints that track the rotations between joints. These sensors make it possible to geometrically calculate the position and rotation of the object. These devices are generally cheap and track changes fast, but the usable area is constrained by the arm length and the mechanical parts will wear out over time.

Electromagnetic trackers use transmitters and receivers to sense how the tracked objects are oriented and positioned. While mechanical devices might be stiff enough that the user can let go of the object, electromagnetically tracked devices have to be kept in hand at all times when tracking. They generally use less desk space than the mechanical counterparts and the tracking area can be made larger with added receivers. However, they are prone to interference from other electrical devices or magnetic objects. [13]

Optical devices are tracked with cameras. The devices can have some easily recognizable markers, such as infrared LEDs, so the system has to do less calculations when trying to recognize the tracked object. Vision based systems are talked about more in section 3.2.4, but in short they generally have large operation areas, but their greatest pitfall are occlusion problems. If any other objects, such as the user's hand, block the camera from seeing the object or its markers, the tracking halts and the system has to predict the motions or simply tell the user to bring the object back in sight.

Acoustic systems do not differ greatly from electromagnetically tracked systems. They use sound emitters and receivers instead of electromagnetism. They also get interference, but this time from echoing sound waves from other surfaces. Physical objects between the tracked object and trackers cause problems, just as with optical systems. [13]

Intertial tracking systems use devices such as gyroscopes to sense the object's motion internally. This means no wires in the way, as long as the data transmission is done wirelessly. Although the setup is easy, gyroscope based devices start to drift over time and need calibration according to the surrounding temperature. [13]

When compared to a computer mouse, using these integral 6 DOF devices generally results in faster task completion times and faster learning, because moving one's hand is a natural way for users to positions objects. On the other hand, accuracy is not always on par with the mouse or knob based systems and if the devices are free moving, fatigue becomes a problem. Holding one's arm up and moving it around in the air takes more effort than using a computer mouse on a desk. [20, 38]

It might feel like 6 DOF controllers should always be used instead of a traditional computer mouse, since they are supersets of 2 DOF devices. A controller with 6 DOF can do everything a mouse can and more, while a mouse is generally confined to 2 DOF. Usually reasons boil down to ergonomics, cost and accuracy. Modern mice are usually quite light, cheap and are offered in various shapes fitting practically any size of a hand. 6 DOF devices might require a lot of desk space, can be quite costly due to being specialized equipment and might not be as gentle on the user's arm when being used for 8 hours per day or longer.[18] Specialized hardware also requires support in the software. For an input tool to become popular, it needs software support, but on the other hand implementing support for it costs time and money. If the device is very uncommon, software vendors may feel like it is not worth the money to add support for it.

3.2.4 Computer vision based input

A very natural way to create pose and motion data is to use physical dolls or living actors to act the poses and motions in front of cameras. Stop motion animation has been used and is still used to make feature length movies. Stop motion animation involves moving and molding objects incrementally and photographing each frame separately. When the frames are played back sequentially it looks like the inanimate objects are moving. When computers are added to the equation, the computer can take fewer key frames and interpolate the frames between them. This makes animating faster and the amount of frames between key frames can be adjusted so the fluidity of the animation does not depend on how many photographs of the incremental motions are taken.

Feng et al. created an artist doll based capture system [10] that uses two cameras to look at a wooden doll that has its joints painted in different colors. The computer calculates from the photographs how the doll is posed. Even though this particular system is intended for searching a database of already existing motion data, the same idea can be used to create key frames for an animation.

Advantages of these doll based systems are that they do not generally require a lot of space so a desk next to a computer is sufficient. Neither do they require much of physical effort from the animator, as long as the doll is made of a material that is easy to manipulate. Using materials such as wax to make the doll makes it possible to create quite imaginative characters, so the tracked skeletons are not limited to humanoid forms.

When it comes to computer animation, one particularly successful computer vision based interaction technique is motion capture. Motion capture has become one of the most common ways to create fluid and lifelike animations. They are widely used when producing computer games, animated films or live action films with added computer generated elements. When using motion capture, an actor or multiple actors act in front of camera, their motions are tracked and are applied to virtual characters in real time (online) or at a later point (offline). These capture systems range from single-camera setups that look at the actor from one angle to multi-camera setups where the actor is at all times seen from multiple angles to combat occlusion problems and not to confuse the actor's limbs.

Motion capture systems can be divided into two categories: with or without markers. When using systems with markers, the actors wear bodysuits that have active electronic markers that may flash in certain order or passive reflective markers that are seen at all times unless occluded by something. Both types of markers are put in predetermined places that mark certain spots on the skeleton on which the motions will be mapped. The computer software uses these markers and their relative positions to deduce which limbs move where and how the motions should look like on the character.

Markerless systems do not require the actors to wear suits with markings or to put any such markings on their clothes. The system analyzes the input from one or multiple cameras and tries to build a skeleton based on its knowledge on human form. These are more suitable especially for video games, where suiting up to play a game can quickly become cumbersome. Markerless systems are faster to start using, but the tracking accuracy isn't as good as with passive or active marker systems [24].

Since the actors are most commonly human, their motions sometimes have to be mapped to remarkably different skeletons than ours. For example an animated film might have aliens with arms twice as long as a human, while still have tiny legs and torso in comparison. Without some sort of animation retargeting the human motions will not produce the wanted effects when applied to the alien body. Luckily this is a widely researched field and there are systems such as KinÊtre by Chen et al. [7] that can map human motions to objects such as chairs or work by Ishigaki et al. [16] that allows real time control of video game characters.

Motion capture technology can also be used on facial animation. As the actors talk and make faces to convey emotion, the markers on the actor's face move and the tracked motions can be applied to virtual faces just as full body capture is applied to skeletons. Although motion capture generally is not used to capture the body type or appearance of the actors, because of facial capture an increasing amount of video game characters are being modeled after the actors who do their motion capture performances and voice acting. This way the end result can be as close to reality as possible.

Although motion capture produces very life-like animations, it is not without downsides. The equipment cost is high especially when using multi-camera setups, the performance space tends to be very large and they require actors to perform the motions. Actors introduce the problem of mapping human motions to different skeletons.[17]

Common issues with all computer vision based systems are lighting, camera placement and calibration.[17] If the system uses light visible to the human eye, same lighting based problems are present as in photography. Not enough light and the system can not see enough to discern between limbs and on the other hand too much light causes overexposure and the view is again obscured. Some systems use infrared light that comes from an infrared projector near the camera and reflects from the target surfaces. These also have problems with lighting, since some light sources like the sun and incandescent light bulbs radiate a lot of infrared light that is often in the same wavelength range as the infrared light from the projectors.

Since cameras can not see through solid objects, camera placement becomes a thing to consider. Some angles are better than others when trying to maximize how well different limbs can be tracked visually. Bad camera angles may cause body parts to occlude other body parts too much for the computer to handle. The amount of cameras has to be considered as well. The more cameras you have, the more angles you can cover, but also the equipment cost goes up proportionally.

Camera systems usually have to be calibrated in some way. Some systems need to see the scene without the tracked objects so the system knows how the background looks like and can ignore it while analyzing the data. Some stereo systems like Microsoft Kinect have to be calibrated so the depth data comes out right. Motion capture systems with multiple cameras have to be calibrated so the camera positions and tracking area is known to the system.

3.2.5 Physically manipulable devices

Physically manipulable devices that track the joint rotations and positions electronically do not have the occlusion problems that are present in camera based systems. The devices have internal sensors that keep track on how the skeletal structure is posed and the computer can map this data to different characters much like camera based motion capture systems do. The ease of use of these devices is comparable to vision based dolls, since only the tracking system changes. Also, the electronics provide some extra features that can not be done with simple dolls in front of cameras.

One interesting feature in incorporating electronics into physical skeletons is that the joints can have motors in them. This makes it possible to make the physical skeleton take a preconfigured pose quickly without the user manually resetting it to a starting pose or a existing key frame. One such device is an actuated physical puppet by Yoshizaki et al. [37] On top of the automated pose reconfiguration, the motors can make the puppet make more humanlike motions when for example reaching with an arm towards a target. Some joints might resist change because human joints have limited ranges of motion and some body parts can be rotated by multiple joints, such as hand twisting can be done from the shoulder or from the elbow. With recorded human data the motors can make the redundant rotations more realistic.

Because such devices are quite complicated, the skeletal structure is generally fixed and can not be easily changed to imitate virtual characters with significantly different skeletons. [19] Of course the joints can be mapped to the virtual character's joints even if the distances between joints are different, but then the results have to be constantly observed on a screen, because the physical model does not represent the virtual character's poses anymore.

In contrast to such fixed topology devices, Jacobson et al. have made a modular input device [19] that consists of interchangeable, hot-pluggable parts that can be put together to form different skeletons. The parts can be of different lengths and shapes and can have multiple connectors, so branching is also possible. The system automatically notices changes in the topology and

the mapping to a virtual character is done semi-automatically. Compared to mouse and keyboard posing, this approach did not have significant difference in completion time and the accuracy of the final pose, but the amount of work needed was clearly lower.

Physically poseable dolls and devices, no matter if they are tracked by vision or internally, always have the advantage of great naturalness. The users can look at the devices from different angles and rotate them around with their hands as long as the devices are handy enough. Physical feedback from touch is there and as long as the topology between the physical device and the virtual character are close enough to each other, the users do not have to look at a computer screen all the time to see the if the pose is getting along.

3.2.6 Wearable devices

Two early wearable devices with 3D capabilities were Z-Glove and DataGlove. Both of the devices are gloves that have positional and rotation tracking in 3D, tactile feedback in fingers and flex sensors, that track how much each finger is bent. Manipulating objects with such devices is very natural, but the devices are now roughly 30 years old and quite dated. [39]. A more modern version of such a glove is for example the setup by Gallo [12] that combines a DG5 VHand 2.0 glove with a Nintendo Wiimote wireless game controller. With this setup, the user can grab objects, translate and rotate them and control the camera to object distance. The device was intended for exploring 3D medical data, but one can see how such a setup could also work when posing an virtual character for an animation.

Since the wearable devices have direct contact to multiple parts of skin or clothes, they can add one feature that is missing from many other interaction devices, which is tactile feedback. For example a glove can give some sort of stimulus to fingers that are in contact with a virtual object. If the glove had some motorized joints and a somewhat stiff skin, the glove could even resist grabbing motions and make it feel like the object is really in the user's hand.

Obvious drawback to using wearable devices is that the user has to put the device on before use. Depending on how complicated and cumbersome the device is, this could be just a slight annoyance or a major issue. If the setup process of such a system takes so long that it is sensible to keep it on for the whole day, then doing animation with it would pretty much tie the user or users to the task for the whole duration.

With the advent of modern head mounted display (HMD) devices, more and more wearable devices are coming to the market to enhance the immersion of those HMD devices. Many of them might have animation potential, but little research has been done yet. Nevertheless, Thomas Zimmerman said it best: "*Just as speech is our natural means of communication, the human hand is our natural means of manipulating the physical world.*" [39]

3.3 Output devices

Since animation is such a visual field, using computers to generate animations without any visual feedback would be quite frustrating. The visual feedback devices essentially come in two flavors, 2D and 3D display devices. In this section we go through some common devices in use and compare how well they fare in 3D animation.

3.3.1 Monoscopic displays

As of now, most desktop computer and laptop monitors are only capable of producing a monoscopic 2D image. If we want to present a 3D environment on such a flat display, the 3D points need to be mapped to a 2D plane. This process is called 3D projection. There are different types of projection, such as orthographic projection and perspective projection. Orthographic projection ignores such perspective effects as objects in the distance looking smaller than ones close to the viewpoint, so it is not feasible for 3D posing and animation. The user has no idea which objects are close and which further away, unless they are occluded by other objects.

On the other hand, perspective projection strives for a realistic presentation of the world. The view is similar to what you would see if you looked at a view with one eye open. You lack depth because of stereo vision, but you can still have some sort of assumptions on how far the objects are based on their size and how they are positioned compared to other objects.

Even if the perspective projection is trying to be as realistic as possible on a flat surface, trying to grab an object with a 3 DOF or 6 DOF input device while looking at the 2D perspective projection is analogous to trying to grab an object in real life with one eye closed. On top of that, many input devices lack any sort of tactile feedback, so the user can not count on feeling the object either. This obviously poses a problem when trying to pose a 3D character.

Research shows that 3D tasks such as positioning and pointing are slower when using monoscopic display devices and 6 DOF input devices, but there is no significant disadvantage when doing rotations [4, 5]. On the other hand, object selection works quite well on 2D screens, when using a 2 DOF input device such as a mouse with ray casting [25, 26].

In conclusion, monoscopic display devices are well sufficient for object selection when using 2 DOF input devices, but object manipulation is more cumbersome than with stereoscopic display devices.

3.3.2 Stereoscopic displays

Stereoscopic displays use various techniques to display different information for both eyes. These images are combined in the human brain and as a result user gets depth perception. This way the user does not have to settle for a 2D perspective projection, although both eyes do get their own perspective projections when using a flat display device to provide the image data to the eyes.

The displays either show the images simulatenously to both eyes or use techniques that make it possible to alter which eye is going to receive the information. Movie theaters, some televisions and monitors use either active shutter glasses or passive polarizing glasses. Active shutter glasses are synchronized with the display device and switch at high rates which eye can see the images. When the shutting rate is high enough the vision system considers the input as stereoscopic and the user can see depth. Problem with the glasses is that they generally make the display slightly darker than it would be without using them. Moreover, they decrease the viewing angle. If the user is not watching the screen from a suitable angle, the image quality plummets.

Passive polarization glasses have one lens with horizontal polarization and the other one with vertical polarization. The display device has filters that make it possible for the glasses to show every other frame for the other eye. They do not require electricity and have better viewing angles than their active counterparts.

One widely researched field in stereoscopic displays are head mounted displays (HMD). These devices are helmets or goggles that display imagery simulatenously to both eyes even if the users turn their head to either side. They often have some sort of head tracking systems in place, so the user's head motions change his or her viewpoint in the virtual world. Possibly the

first HMD VR system was the Sword of Damocles by Sutherland [32]. The device was quite cumbersome and the tracking features required a considerable space above the user, where a tracking system hanged above the user, hence the name. The upcoming devices had improvements, but were still quite bulky to be used for extended periods of time [27].

HMD devices have had a boost in popularity lately with the introduction of more recreation oriented HMDs like the Oculus Rift by Oculus VR and Vive by Valve Corporation and HTC Corporation. They are much lighter and compact than earlier designs and can be worn for a longer time before the user gets fatigued by the weight. These devices have an enclosed screen that is positioned in front of the user's eyes by using head straps. Between the screen and the user's eyes are lenses that increase the field of view (FOV). Because the lenses distort the viewed image, the image on the screen has to be distorted as well to counterbalance the lens distortion. This is why the image looks rounded and oddly colored on the edges if looked at on a regular computer screen. Pictures of this distortion can be seen on the edges of figures 5.1 and 5.7a.

Both of these modern devices track the position and rotation of the device, so the user can look at objects of interest from different angles by moving his head or upped body. This way actions like camera rotation and movement can be done without having to use the hands that are possibly using manipulation tools.

Research shows that having stereoscopic view of the scene means faster completion times when pointing [4] and positioning [5] in 3D environments, but not when matching object rotations [5]. In neither study was head tracking found to decrease task completion times, but for camera control purposes the possibilities are intriguing.

Chapter 4

Environment

In this chapter we introduce the hardware and software used in this project.

4.1 Hardware

The hardware needs to be able to show the user 3D imagery, follow the user's head motions and track the user's hands. These requirements are met by combining an Oculus Rift DK2 head mounted display (HMD) and Leap Motion controller.

4.1.1 Leap Motion Controller

The Leap Motion controller is a hand and tool tracking device developed and manufactured by Leap Motion, Inc. The controller uses infrared to track hands and tools within an effective range of 0.25 cm to 60 cm. The tracking angle is 150° wide and 120° deep. The device weighs 45 g and has measurements of 13 mm × 13 mm × 76 mm.

The tracking device was originally designed to be placed on a flat table surface and the users would move their hands above the device. After the Oculus Rift and other HMD devices gained popularity, Leap Motion released a VR Developer Mount that is attached to a VR-device with double sided tape and the Leap Motion controller can be placed in a slot in the mount. This way the tracker does not have to be permanently attached to the VR-device and can be still used in desktop mode if needed. A Leap Motion controller attached to an Oculus Rift DK2 can be seen in figure 4.1. Due to

the contoller placement, the hands can be only tracked when the users have them in view. If the users turn their heads and keep their hands stationary, the controller loses the hands.

Figure 4.1: A Leap Motion controller attached to an Oculus Rift DK2

The Leap Motion controller has some physical limitations due to the way how it works. Two CMOS sensors in the device track infrared light with the wavelenght of 850 nm[9]. This somewhat restricts how the room where the device is used can be lit. Light sources that have high power in the infrared spectrum near the tracked 850 nm degrade the quality and reliability of tracking. Light sources that usually cause less than ideal conditions are incandescent light bulbs, halogen lamps and daylight. LED lamps and fluorescent lamps generally cause less problems with tracking. Leap Motion has a setting called *Robust Mode* that tries to improve tracking in a brightly lit environment, but currently it can not completely negate unfavorable lighting conditions.

The sensors sense infrared light originally emitted from the three neigh-

boring infrared LEDs and are reflected from the users hands. If the area where the controller is pointed has highly reflecting surfaces the infrared light can be reflected back to the device further away than intended. This may trick the device into believing the reflecting surface is actually a hand or the reflected light can cause problems with hand tracking, because some of the light can shine back between the users fingers. For best results the facing direction should not have highly reflecting objects as mirrors, glossy computer screens, brightly reflecting white walls or metallic objects such as lamp shades.

This project uses Leap Motion controller firmware version 1.7.0 and SDK version 2.2.5+26752. The "Known issues list" for this version includes reduced pinch gesture tracking in HMD mode when compared to the standard desktop mode, which is a bit problematic, since holding the characters body parts by pinching comes to the users naturally. Some people had more issues with pinching than others and we included a secondary way of grabbing the body parts. Known issues list also includes overall reduced tracking quality in HMD mode compared to the desktop mode, but the quality is gradually getting better.

4.1.2 Oculus Rift Development Kit 2

Oculus Rift DK2 is a VR HMD device developed by Oculus VR. It features a 1920×1080 low-persistence PenTile AMOLED display with 75Hz refresh rate, which means each eye gets a resolution of 960×1080. The device has 6 DOF tracking with rotation tracking done internally and positional tracking done with a near infrared CMOS sensor. The device has lenses between the user's eyes and the screen, which gives the device a 100° nominal FOV. The device connects via HDMI 1.4b and USB 2.0. The device weighs 440 g.

The device shows each eye simulatenously a separate image through lenses, that give a wider FOV than it would be without them. Because of the distortion caused by the lenses, the original image needs to be distorted as well to compensate for this. Due to this, simply using the Oculus Rift as a secondary monitor and displaying the computer desktop on it will not work, because the image has to be processed before it can be seen properly through the lenses.

The 6 DOF tracking means the users can rotate and move their heads and the virtual world follows, as long as head tracking features have been implemented in the software. The users can for example tilt their heads and peek through windows. Although research shows that head tracking does not

make manipulating objects faster when using a stereoscopic display device [4, 5], the head tracking can be used to control the camera. With Oculus Rift, the user can move his or her head and upper body to look at the object of interest from different angles without having to use another interaction device to move the viewpoint. This makes making small adjustments to the viewpoint extremely fast and the way to move the viewpoint is very natural to human beings, who in real life can do the same when handling physical objects. In the end, being able to move around the object makes the manipulation faster as well, since the character's arm might be occluded by the body and a slight adjustment in the user's position makes it visible instead of rotating the object and then possibly back to its original position.

When using a Leap Motion controller in head mounted display (HMD) mode with an Oculus Rift, the Leap Motion controller is attached to the front panel of the Oculus Rift device with either glue or tape. This configuration can sometimes cause problems with positional head tracking, because the positional infrared camera has to see the infrared LEDs in the Rift's front panel and sides. While the Leap Motion controller itself does not block the view to the LEDs when installed correctly, the user's hands can sometimes block enough of the view so the positional tracking loses the headset's position. The user can see this as jerky movement of the whole virtual world and such unexpected movements can result in some discomfort. To minimize this, we found that turning about 45° away sideways from the monitor mounted camera keeps hands from blocking the LEDs too much. The camera itself can also be placed on a table in such a way that it always sees enough of the LEDs.

4.1.3 Wireless Xbox 360 Controller

The *Wireless Xbox 360 controller* is a gamepad manufactured by Microsoft Corporation for the Xbox 360 video game console. The controller is also compatible with Windows Vista, Windows 7, Windows 8 and some other operating systems with the aid of a *Wireless Gaming Receiver* also manufactured by Microsoft. The controller has these inputs

- 2x analog control sticks
- 2x analog triggers
- Digital D-pad
- 11x digital buttons:

- 4x face buttons (A, B, X, Y)
- 2x shoulder or 'bumper' buttons
- Back and Start buttons
- 2x digital buttons that activate when the analog sticks are pressed down
- Guide button

In this project the controller was used to navigate in the virtual space and perform some of the actions the user can also do with hand tracking. This way the users have a choice if they want to do everything with tracked hands or keep one hand on the controller and only use the second hand for posing. Not all of the inputs are utilized and the button layout can be seen in figure 5.6.

4.1.4 Keyboard

In this project the keyboard served two purposes: application control (reset and exit) and as an alternative grabbing tool.

Possible other purposes for the keyboard would be expanded application control tasks, such as loading different models, saving and loading animation data and exporting data to different formats. As of now, the keyboard usability is limited.

4.2 Software

The selected software must be able to take advantage of SDKs offered by Oculus VR and Leap Motion. Additionally, the wider support for 3D manipulation the software has, the less development time needs to be spent on the engine and presentation side and more time can be allocated in making the user experience natural.

4.2.1 Unity

Unity is a cross-platform game engine developed by Unity Technologies. It was originally released in 2005 and has since grown into one of the most used

game engines. Scripting in Unity uses Mono, which is an open source implementation of Microsoft's .NET framework. Scripting can be done in either C#, Boo or UnityScript. UnityScript is syntactically similar to JavaScript and Unity Technologies even calls it JavaScript in the integrated development environment (IDE) and in the documentation, but there are some differences. This project was done completely in C# using the Unity version 4.6.1f.

Unity was chosen for this project because it is officially supported by both Oculus VR and Leap Motion. While Oculus VR offers integration for other engines such as Unreal Engine 4, at the time of writing Leap Motion support for Unity is better than for any other relevant platform.

Oculus VR and Leap Motion provide ready made assets for Unity. The most important assets are prefabricated camera rigs with hand controllers included, so the scene can be rendered properly with the Oculus Rift device and the hands detected by the Leap Motion are rendered in the correct position related to the user's head. Leap Motion assets also include different looking skeletal models to represent tracked hands and a few different hand models users can expand with their own code.

Chapter 5

Implementation

This chapter describes the system that was implemented through iterative prototyping and user testing. We organized three testing rounds with four, four and five users respectively. A couple of differences between the prototypes and the final version are illustrated in figure 5.7. An overview picture of the final version is shown in figure 5.1.

5.1 System overview

The basic workflow goes like this: as the system boots up, the users take comfortable positions on their chairs and set themselves up so their hands will not block the Oculus Rift positional tracking camera and there are no reflective surfaces in front of them, so the Leap Motion controller does not get interference. When the users are ready, they should press the Back button on the Xbox 360 controller or Tab on the keyboard. This resets the head tracking and puts the viewpoint in the default position in front of the character. Now the system has been set up.

The character starts in a standard T-pose and the timeline is set on the leftmost end, which marks the time at zero seconds and the initial keyframe has been created. Now the user can pose the character into the initial pose he or she wants. This is done by manipulating the character with the Leap Motion pinching gesture or using the Xbox 360 controller or keyboard if the user wants to use the button-to-grab mode. The user can move around the character with the Xbox 360 controller or with head motions. When the user is done with the initial pose, he or she should move the timeline slider to another position on the timeline and start posing again. As soon as the

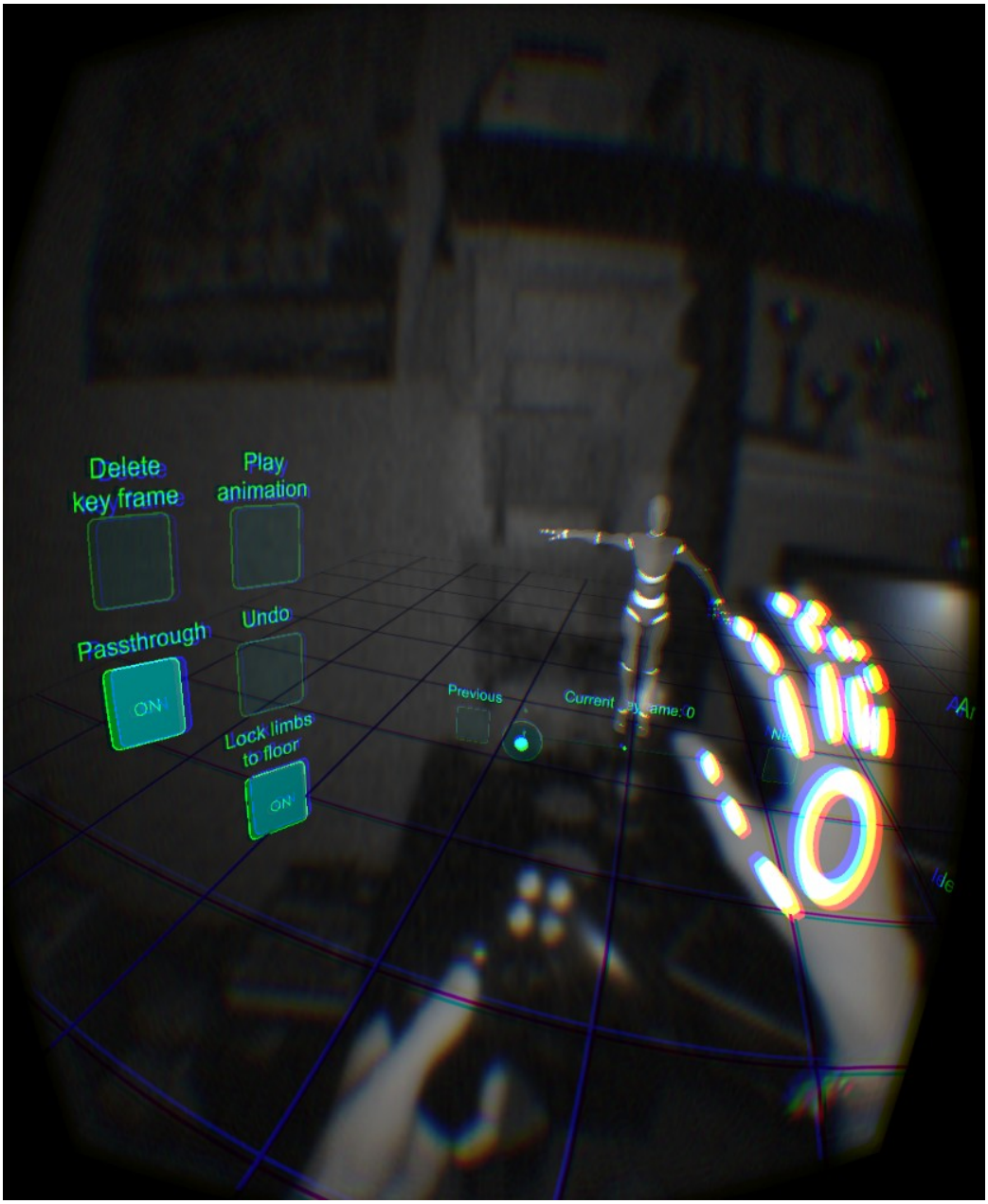


Figure 5.1: Overall picture of the tool. The user is using the Xbox 360 controller with one hand and posing the character with the other. To see the buttons on right the user has to turn his or her head.

user grabs and lets go of the character in this new timeline position, a new keyframe is created. Then the user can do additional adjustments to the pose and the new keyframe is edited. The user should repeat these steps until the

wanted amount of keyframes are created.

If at any point the user thinks one of the keyframes does not fit the flow, the user can select it with the timeline slider or by using the "Previous/Next Keyframe" buttons and push the delete keyframe button. On the other hand, if some part of the animation needs additional keyframes between existing ones, the user can simply go to that part of the animation and pose normally. The new keyframe is created between the old keyframes. The user can also use undo to cancel unwanted changes to a pose, but the undo buffer resets every time the user moves in any direction on the timeline or changes between keyframes with the buttons.

While the user is doing posing work or after the animation is ready, he or she can push the "Play Animation" button to see how the animation looks like. If the user feels like the animation is going too fast or too slow, he or she can adjust the animation length with the buttons on his or her right side or the D-pad on the Xbox 360 controller.

Because the Leap Motion controller works better the more it can see of the user's fingers, the way you keep your hand while grabbing makes a big difference on the tracking reliability when using the pinching mode. Although the gesture can get tiring on the hand after a while, it is best if the user can keep all the other fingers extended and visible while grabbing with the thumb and index finger. A proper way is shown in figure 5.7a.

5.2 Inverse Kinematics

There are two ways how a model and its limbs can be posed: *forward kinematics (FK)* and *inverse kinematics (IK)*. If we have a model with an arm and take a chain of bones and joints from the shoulder joint to the end of the index finger, with forward kinematics we can calculate the position of the index finger from the lengths of the bones and angles of the joints. With inverse kinematics we can calculate the angles of the joints when we know the desired end position for the index finger and the lengths of the chain's bones. In short, FK poses the chain from root to tip and IK posing is based on the target. Since in this project we want to be able to pose the character by using the Leap Motion tracker and grabbing the model's limbs in a virtual space, inverse kinematics is the proper approach.

Unity has a built in inverse kinematics solver, but it was a part of the Pro-only features until the release of Unity 5. As this project was developed

using Unity version 4.6.1, we had to implement the solver by ourselves. Our solver is a C# implementation of *Non-Iterative, Closed-Form, Inverse Kinematic Chain Solver (NCF IK)* as described by Philip Taylor [33]. Compared to other inverse kinematics solving methods such as the Jacobian Inversion Method [35] or the Cyclic-Coordinate Descent Method [36], the NCF IK can be solved in one pass with no iteration and is at least in our opinion simpler to implement. NCF IK does not support twisting the target limb to a correct position, but this does not matter in this project since the twisting will be done by the user manually. The implementation can solve IK chains of length N. The solver tries to preserve the initial shape of the chain when reaching for the target.

Pseudocode for the solver can be seen in program 5.1. In this pseudocode the angles and targets are as follows:

Chain target alignment
: Each bone is aligned towards the target before determining what to do based on the bone's location in the chain. This is done by aligning the chain by making the vector from the current (sub)chain's root to the tip of the chain parallel with the vector from the current (sub)chain's root to the target. This maintains the overall shape of the chain and keeps all deformations on a single plane.

IK bone angle
: Angle between the bone length vector and the bone to goal vector.

FK bone angle
: Angle between the bone to chain tip vector before solving and bone length vector.

Maximum FK bone angle
: The angle between the bone length vector and the vector from the current (sub)chain's root to the tip of the chain, if the rest of the chain was laid out straight with the tip of the chain staying where it was before straightening. This tells how much the bone can rotate away from the current tip of the chain while still being able to reach it with the rest of the chain.

Maximum IK bone angle
: Same as the above, except the tip of the chain stays at the IK target. This tells how much the bone can rotate away from the IK target while still being able to reach it with the rest of the chain.

```
Function SolveIKChain( chain )
begin
    calculate chain target alignment

    for each bone in chain
    begin
        apply chain target alignment to bone

        if bone is last bone
            aim bone at target
        else if bone is second last
            use trigonometry to calculate bone angle
        else
        begin
            determine FK bone angle
            determine maximum FK bone angle
            determine maximum IK bone angle

            IK bone angle = ( FK bone angle / maximum FK bone angle ) *
                            maximum IK bone angle
        end
    end
end
```

Program 5.1: Pseudocode for NCF IK [33]

The solver itself resides in the poseable character's behavior script called *PoseableCharacter*. The solver takes a *ChainData* object as a parameter. The ChainData contains the chain's starting body part (for example left shoulder), ending body part (for example the left forearm) and the end point (for example left wrist, as that is where the left forearm ends). These are stored as Unity Transforms, which all objects in the scene have. These transforms are the transforms the character's limb objects have. These are used to calculate the bone lengths and to rotate the character's limbs, as any rotation to the transforms apply directly to the character. The ChainData object also includes the target position, all the transforms in the chain as a list, the lengths of the bones in a list and the total length of the chain. All this information is needed by the solver and the ChainData is built by a method in the PoseableCharacter script that takes the above mentioned three body part transforms as parameters.

The starting body parts for the chain are set by the user in Unity editor. For arms and hands the starting part is now in the corresponding shoulders and for legs and feet the starting parts are right and left sides of the hips. If the system at some point has a possibility to import models inside the system and not the editor, this feature needs to be changed so the editor is not needed.

5.3 Leap Motion controller

The development for Leap Motion controller features involved downloading the Leap Motion's Unity Core Assets package and modifying the relevant classes and creating some new ones.

5.3.1 Hand tracking

The basis for the hands are the *RigidHand* prefab and class found in the Leap Motion's Unity Core Assets package. RigidHand, which inherits from *SkeletalHand*, handles moving the hand model in the scene based on the Leap Motion controller's input. The hand model itself is made out of various polyhedra and the bones in the fingers are not visibly connected to each other, the palm has a see through hole in the middle and there are no visible wrist or forearm models. This hand model was chosen over models that are more realistic human hands with attached arms, because the model covers less of the view when in use while still being very hand-like and natural. The Leap

Motion's Unity assets include plenty of *RiggedHand*s that are modeled after real human hands and arms, and they are better suited for virtual worlds where a different sort of immersion is sought-after. The hand model can be seen in figures 5.1 and 5.7a.

Pinching gesture, grab-by-button and rotating the character's limbs are features we added to the RigidHand behavior. The pinching gesture checks whether the thumb and index finger's finger tips are close enough to each other. If they are and no grabbing has been initiated yet, the script checks if there are any body parts close to the mid point between the fingers. If there is a body part, a sound notification in played, a translucent sphere is created on the surface of the body part to show where it was grabbed and a ChainData object is created, unless the grabbed body part was the character's hips. If the part was hips, then all translation and rotation of the palm of the hand is directly applied to it. If ChainData was created, any rotation of the palm of the hand is applied to the starting body part of the chain. If the user grabs the character's left foot, all rotations are applied to the joint that connects the left thigh to hips. After the rotation has been applied, the system solves the IK chain and rotates joints accordingly. The IK target stays between the pinching fingers when the hand moves around. The pinching gesture ends when the distance between index finger and thumb go over a set threshold. When the pinching gesture ends and there is an existing keyframe at that time in the animation, the pose stored in that keyframe is edited. Otherwise a new keyframe is created and the pose is stored in it.

If the button to grab a limb is pressed, same things as with pinching gesture happen, except the point where the limb is selected is at the tip of the index finger. Hand movement and rotation are applied as with the pinching gesture.

There is no limit on how many hands can be used simultaneously, so the user can have a friend to "help" with the animation tasks, but the tracking area quickly gets cluttered and the other user will not have a 3D view of the scene anyway, so the whole idea is quite useless. The only way left and right hands differ in the system is that the right hand gets a red sphere as the grabbing marker and the left hand gets a blue one. This is useful if two body parts are grabbed and the grabbing positions get close to each other.

5.3.2 Image passthrough

With the Leap Motion SDK it is possible to access the infrared camera data. This makes it possible to feed this data into the Unity scene in front

of the Oculus cameras. Consequently the user can see whatever the Leap Motion sees and this is quite useful if the application requires the use of input devices such as a mouse, a keyboard or a game controller. Without the image passthrough mode, the user might lose track where the mouse or keyboard is while being immersed in the virtual world. The image passthrough feature requires that the *Allow Images* setting is enabled in the Leap Motion control panel. A comparison between the view with image passthrough enabled and disabled is in figure 5.2.

The image data is rendered on two quads, one for each eye, that are hierarchically children of the Oculus Rift camera rig prefab. We made the quads togglable by a button, that can be pressed with Leap Motion tracked hands. The mode is not togglable by keyboard or Xbox 360 controller buttons. When the button is in OFF position, the quads are simply not rendered.

By default the passthrough is on, so the user can see where the Oculus Rift positional camera and other input devices are while he or she is settling into a comfortable position. Some people prefer to keep the passthrough on while posing the character, while some find it distracting and turn it off. It also causes some discomfort to some people, since the image data is slightly delayed compared to head position and orientation changes. The difference is not much, but it is easily noticeable if the user turns his or her head left and right repeatedly.

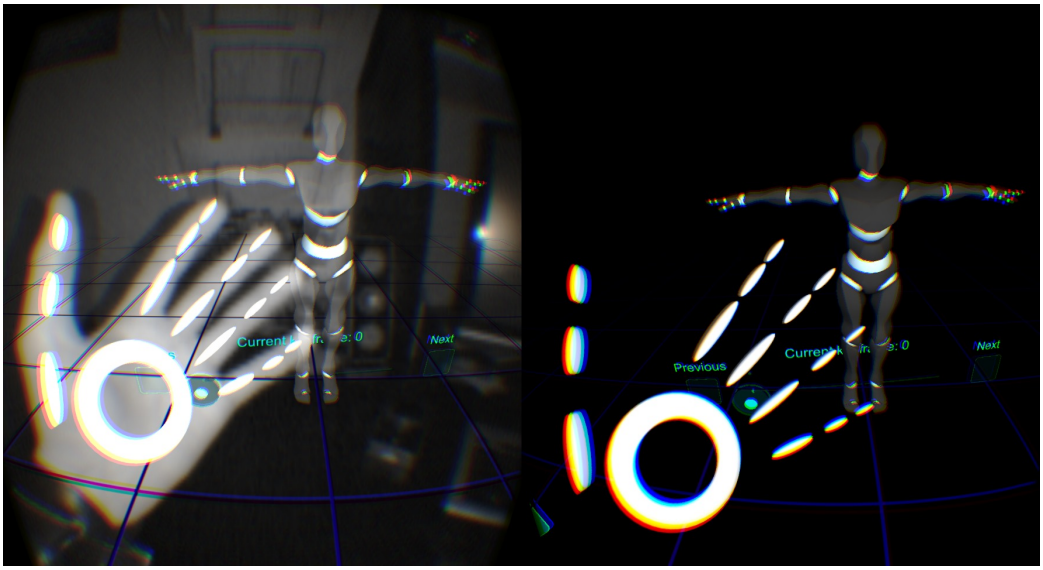


Figure 5.2: Leap Motion image passthrough ON and OFF

5.4 Oculus Rift

Although Oculus Rift provides its own Unity assets, for this project the assets came from Leap Motion. The Leap Motion VR assets combines the Oculus' VR assets with Leap Motion's own. The main difference between the HMD prefabs is the added Leap Motion tracker between the eye positions. With this positioning the tracked hands are rendered in the correct positions in the virtual space related to the user's head.

The user can reset the Oculus Rift position by pressing *Tab* on the keyboard or pressing the *Back* button on the Xbox 360 controller. This resets the initial position for the user in the virtual space, puts the user interface to his or her left and right sides and places the poseable character in front of him or her. Resetting is usually needed in three situations. First is when beginning to use the tool and still looking for a comfortable position that is favorable for the Leap Motion controller and the Oculus Rift positional tracker.

The second reason is the user might have a swivel chair with rollers and he accidentally moves around too much when immersed in the virtual space. Swiveling too much can cause the Oculus Rift positional tracking to fail when it can not see enough of the tracking LEDs. The user can also turn too far away from a table to place the Xbox 360 controller on or use the keyboard. In this situation the user can just return to his or her original physical position and hit the reset position button.

The situation for the third reason can happen if the user accidentally loses the poseable character when moving around the virtual space with the Xbox 360 controller. Instead of flying around and looking for it the user can just hit the reset and the character is again in front of him or her without actually resetting the scene and all keyframes.

In the end the camera rig prefab *LeapOVRPlayerControllerButtonScroll* that includes the Oculus Rift and Leap Motion support was left mostly unchanged. Additions are the widgets for the user interface and the possibility to toggle the image passthrough with a button.

5.5 Poseable character

The character model used as the default testing character in this project is a model called *Alpha* by Mixamo Inc.[23] The character is a generic male

bipedal humanoid and can be seen in figure 5.7a. After importing this model into Unity, the *Ragdoll Wizard* tool in Unity editor was used. The tool creates suitable colliders, rigidbodies and joints for the character. The user has to drag correct limbs from the character's hierarchy to the wizard's properties, which means the tool does not care how many bones the character model has.

The resulting rigidbodies and joints are not needed for this tool to work, so they can be deleted after the ragdoll creation is done. There is no need for physics so the rigidbodies will not do anything and the joints are not used at any point. The colliders are necessary for the body part grabbing and floor to work. When a collider touches the floor, the floor's actions are triggered. The floor is more thoroughly explained in section 5.7. When the grabbing is triggered by pinching or pushing a button, an overlap sphere checks if the there are any colliders inside the radius. If a collider is detected, actions described in section 5.3.1 are performed.

The character has a behavior script *PoseableCharacter* attached to it. The script is responsible for building ChainData and solving IK for the ChainData.

5.6 User Interface

The user interface consists of buttons, text labels and a slider that look like they are floating in space around the user. Hierarchically the buttons and slider are placed as children of the Oculus camera rig. This placement means that after the Oculus Rift tracking position has been reset, the buttons will stay where they are, even if the user moves around on his or her chair. This way the buttons on left and right will come visible by turning the head and the user can get closer to or further away from the buttons if he or she feels like it. If for example the tracking distance deteriorates quickly with distance because of unfavorable lighting conditions the user can move his or her face towards the buttons so the tracking distance from the Leap Motion tracker to the user's fingers gets shorter and the buttons can be pushed again.

5.6.1 Timeline Slider

The animation timeline is shown as a horizontal slider, which has a marker the user can move with his or her hands or an Xbox 360 controller. The timeline shows created keyframes as blue dots above slider. The leftmost

point of the slider is the beginning of the animation and the pose at time zero seconds. The rightmost point is the end of the animation and the pose at user set maximum seconds. The timeline is shown in figure 5.3.

The slider and its behavior script *SliderTimeline* is based on the Leap Motion Unity widget slider that comes with Leap Motion's Unity Core Assets package. Modifications were made so the slider shows dots for each keyframe's position in the animation's time frame instead of showing a set number of dots spread evenly. The largest modification is the scrubbing feature. When the user moves the marker on the timeline, the slider tells the AnimationController (described in section 5.8 to set the character's pose to what it is supposed to be at that part of the animation. It also tells the AnimationController what the currently selected keyframe is.

When the scene has been reset and the camera is located in the default position, the timeline slider is positioned below and slightly further away from the viewpoint than the poseable character is. With this position the user should be able to handle the slider and character without moving around on the chair, since the tracking distance is enough to cover the whole area. If for whatever reason the tracked hands start malfunctioning at the slider distance then the user can physically move a bit forward and move the viewpoint backwards with the Xbox 360 controller so the slider comes closer while the character stays the same distance away from the user.

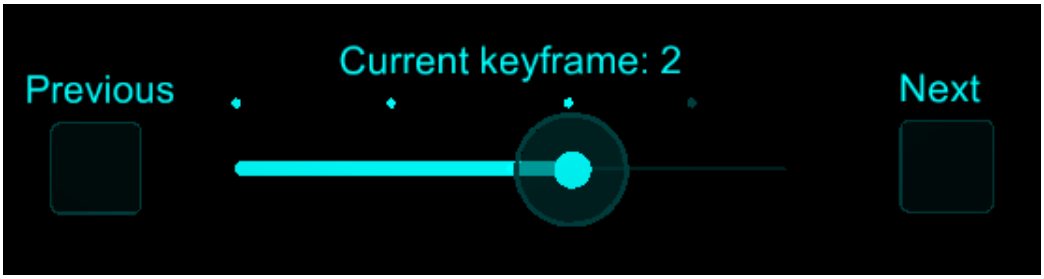


Figure 5.3: Timeline slider with previous and next keyframe buttons

In the first version of the timeline slider the user first had to move the slider marker to a position where he or she wanted to create a new keyframe or edit an existing one, make a pose with the character and then hit a *Create Keyframe* button on the left side menu or hit a corresponding button on the Xbox 360 controller. If the slider marker was set to the end of the animation when creating this new keyframe, a new one second clip was added to the animation and its length was controlled with buttons on the right side of the user. If the slider marker was between two existing keyframes,

a new keyframe was created between them while the total length between the previous keyframes didn't change. This means the new keyframe was an intermediate pose between the original start and end poses and the start and end poses themselves did not change in anyway. Keyframes were removed with a *Remove Keyframe* button or pushing an equivalent Xbox 360 controller button. This removed the keyframe at the slider marker position or the first keyframe on the left from the marker, if the marker was between keyframes.

In user tests this version of the timeline proved to be slightly confusing to the users. Users regularly forgot to push the *Create Keyframe* button before moving to a new position on the slider. This removed all the changes the user had made and no new keyframes were created and no existing keyframes were edited. Sometimes the users were confused if their actions had been saved in any way.

The next and final version removed the *Create Keyframe* button completely. Now the user moves the slider marker on the slider just as before and poses the character. Each time the user lets go off the character a new keyframe is created or an existing one is modified. This removes the need for the user to remember to push the *Create Keyframe* button before starting on his or her next posing action.

In this version the whole animation does not get longer when a new keyframe is created with the slider marker at the right end of the slider. It just marks the end of the animation and the final pose of the character. Instead the buttons that previously set the amount of time between keyframes now control the length of the whole animation. The default setting is 5 seconds. If the user sets the total length as 15 second the leftmost point of the slider is the animation at zero seconds, middle point is 7.5 seconds and the end of the animation occurs at 15 seconds.

The users got a hang of this version faster than the previous one. The users did not have to ask if their actions had been saved and if they can move to an another part of the animation.

5.6.2 Buttons

The buttons are based on the togglable button found in the Leap Motion's Unity Core Assets package called *ButtongToggleBase*. The togglable button is used for the passthrough and floor locking buttons, with only modifications to the logic that tells what the buttons should do when they are on or off.

The rest of the buttons have a new base called *ButtonSingleBase*, that is only on while the button is kept pressed. This type of a button is more suitable for triggering actions rather than changing between modes.

The buttons are used by poking at them with the RigidHand that is tracked by the Leap Motion controller. Any finger or part of the hand model works while pressing the buttons. The buttons have an adjustable depth how far the buttons have to be pushed down before they activate, which means they are not static and have visible depth in them. Some of the buttons have togglable on and off modes which activate and disable features while most of the buttons are single press and perform the binded action each time they are pressed down.

The buttons and their actions are:

Undo
: Undoes the previous posing if possible. A list of previous positions is held as long as the user does not move back or forth on the timeline or does not delete the current keyframe.

Remove keyframe
: Removes the keyframe that is the first one on the left of the current slider position or the keyframe exactly in the current slider position, depending on how the slider is positioned. The keyframe at timeline position 0 can not be deleted.

Passthrough (ON/OFF)
: Toggles the Leap Motion image passthrough mode. When ON the user can see a black and white picture of what the Leap Motion tracker sees. When OFF the area behind the character and buttons is simply black. This button is useful if the user has lost track of the table, keyboard or gamepad. Default state is ON.

Lock limbs to floor (ON/OFF)
: Toggles if the character's limbs stick to the floor. When ON, the character's limbs stay at the positions they first touched the floor. When OFF the limbs slide on the floor as the character moves above the floor.

Previous / Next
: Buttons on the left and right side of the timeline slider. Buttons jump to the previous and next keyframes respectively.

Increase / Decrease animation length
: Increase or decrease the total animation length with these buttons.

This changes the total length of the timeline slider in seconds, but does not change the length visually in virtual space.

The button layout can be seen in figures 5.4 and 5.5.

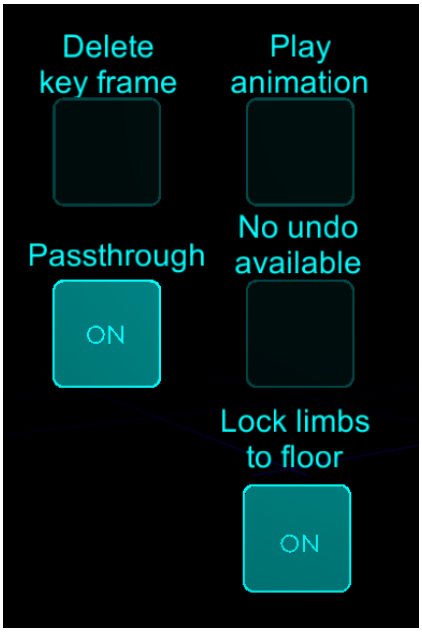


Figure 5.4: Button layout on the left side of the user

5.6.3 Wireless Xbox 360 Controller

The Xbox 360 controller implementation is not complicated. Unity input manager was used to bind buttons for all of the actions described in figure 5.6. Behaviour scripts for floor, timeline slider, animation controller and camera rig check if any of those actions are triggered and act accordingly.

5.7 Floor

The floor is a blue grid that is placed under the character in the virtual world. This floor keeps the character's limbs from going under it, unless the character is grabbed by the hips and forced through the floor. This will cause heavy deformations to its arms and legs. The point of the floor is to show

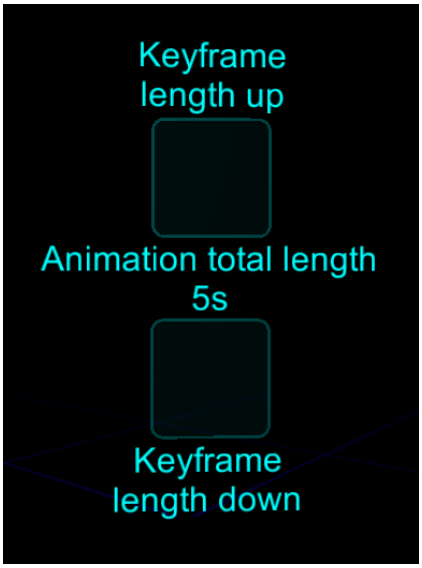


Figure 5.5: Button layout on the right side of the user

the user a plane which the character can walk on and aid with poses like kneeling and walking.

The floor has two modes: lock limbs to floor ON and OFF. The difference between them is if the limb that has touched the floor will stay in place even if its hierarchical parents move it around. If the locking mode is ON and the character's feet are touching the ground, they will stay in place when the character is moved around from the hips. If the mode is OFF, the feet will slide on the floor as the character is moved around. No matter which mode is on, the character's knees will bend if the character is pushed towards the floor by the hips and the knees will straighten if the character is moved away from the floor. Same rules apply to other body parts as well.

The floor does this by keeping track of body part colliders that come in contact with the floor. If a contact happens, a ChainData object is created. This ChainData object is identical to what is created when a body part is grabbed by pinching or pushing a button to grab. In a sense pinching fingers are created in the position where the contact happens. If the locking is on, the pinch will stay in place and if the mode is off, the pinching will follow the characters movements while staying in the floor plane. As long as the contact is happening, the ChainData persists and the IK chain is solved for that body part.

Figure 5.6: Xbox 360 controller layout

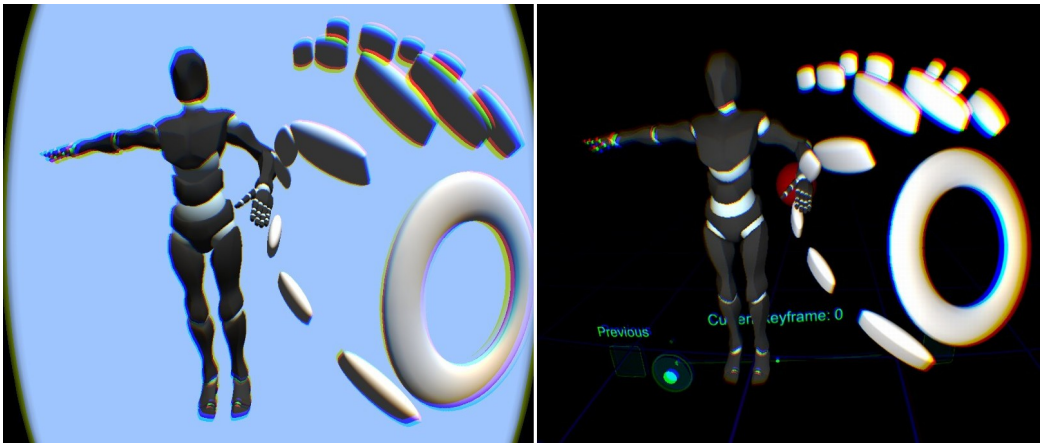
5.8 Animation controller

The *AnimationController* behavior script is responsible for keeping track of the keyframes, undo system, setting the character's pose when changing between keyframes or scrubbing on the timeline and playing the animation whenever the "Play Animation" button is pressed.

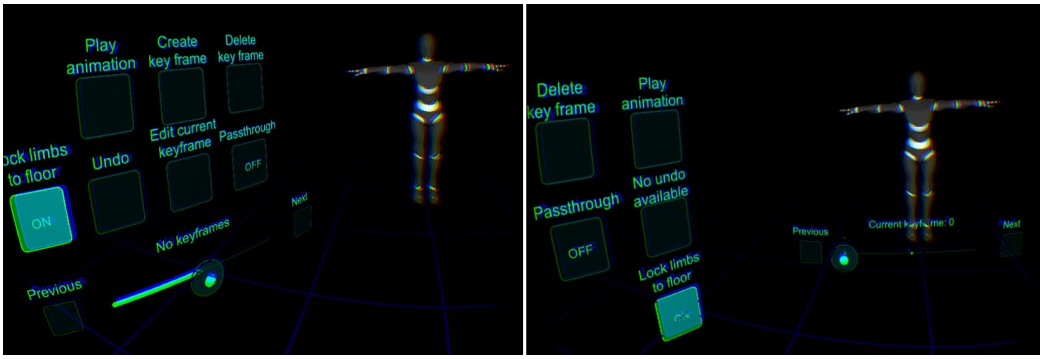
Keyframe data is stored in *Keyframe* objects which the AnimationController has a list of. When the animation is played, the AnimationController interpolates the character's current pose based on the keyframe data, how they are positioned on the timeline and how long the whole animation is.

Whenever the user lets go off the character, AnimationController checks if a keyframe exists at the current position. If there is, then it is updated with the current pose, otherwise a new one is created. The keyframes store their position on the timeline in percentage of how far on the right they are. In the list they are in the same order as they appear on the timeline, so the method that handles animation can go through the list sequentially and use the percentage figures to calculate how long the duration between each keyframe should be.

The undo system stores poses in a list and a new pose is added each time the user starts grabbing. This means the user can cancel his or her previous



(a) On the left the very first IK test with Oculus Rift and Leap Motion controller included. On the right doing the same grab in the final version.



(b) Difference between the original menu design and the final design. The final design is much simpler to use after the removal of Create / Edit Keyframe buttons.

Figure 5.7: The tool in prototype and final phases.

posing or posings by pressing the undo button. The undo buffer is cleared after any action is performed on the timeline.

When the user uses the timeline to scrub, the timeline tells Animation-Controller to set the character's pose to a pose stored in a keyframe or to interpolate a pose between keyframes, if the position on the timeline is between two keyframes.

Chapter 6

Evaluation

6.1 Usability testing

To see how the tool works in real life, a final user test was organized with five professional game industry animators. Three of them had used an earlier version of this tool briefly before and the other two had seen a video of it in action.

The testers were told to familiarize themselves with the controls for five minutes, use 15 minutes to animate whatever animation clip they want and immediately after that fill a system usability scale questionnaire and answer five open questions. After answering the predetermined questions the testers had time to provide any other feedback they might have.

The system usability scale (SUS) was developed by John Brooke in 1986 [6]. It is a simple and fast 10 item Likert scale that has been widely used in usability testing across various technologies. The 10 questions were selected from a pool of 50 questions based on results from 20 testers. The standard 10 questions can be seen in table 6.1. Over 40% of current usability testing is done with SUS. [28]

Of the 10 questions odd numbered questions have a positive tone in their wording while the even numbered questions are worded negatively. To calculate the SUS score, each odd numbered question score is reduced by one (score - 1) and each even numbered score is taken away from five (5 - score). This gives each question a score of 0-4, where zero is always the worst and 4 is always the best result. Then all of the question scores are tallied together and multiplied by 2.5 to get a score of 0 to 100.

Although the scoring is between 0 and 100, the SUS scores should not be considered as percentages. A better way to make sense of the results is to convert the obtained SUS score to a percentile rank using tables based on researched data. A table of such rankings can be seen in 6.2. [28] This way you can find out roughly how usable the system really is. They found out that the mean score is 68, so a system with a SUS score of 68 scores better than approximately 50% of systems.

For the open questions, the testers wrote their answers on a piece of paper immediately after filling the SUS questionnaire and before any debriefing. The open questions were as follows:

- What was good in the tested system?
- What was bad in the tested system?
- What would you change or add?
- If the system would work perfectly, how would you integrate it into your tool palette and workflow?
- For what sort of animation purpose or what work stage this sort of system would work best?

After these questions the testers had time to talk about the system and offer feedback on issues that were not touched upon by the questionnaire and open questions. General conversation about how they liked the hardware and software was had with the testers.

6.2 Usability testing results

The mean SUS score from the five testers is 62.5, which gives the system grade D and places it in the higher end of percentile range 15-34 according to the curved grading scale interpretation of SUS scores by Sauro [28]. The grading scale is shown in table 6.2.

Looking at the average scores per question in figure 6.1, the major score reducers are user confidence when using the system at 1.4 points and user eagerness to use the system frequently at 1.6 points, both on a 0-4 point scale. In the debriefing the testers said the major reasons they had, that decrease their confidence in using the system, were occasionally unreliable

1 = Strongly disagree, 5 = Strongly agree	1	2	3	4	5
1. I think that I would like to use this system frequently.					
2. I found the system unnecessarily complex.					
3. I thought the system was easy to use.					
4. I think that I would need the support of a technical person to be able to use this system.					
5. I found the various functions in this system were well integrated.					
6. I thought there was too much inconsistency in this system.					
7. I would imagine that most people would learn to use this system very quickly.					
8. I found the system very cumbersome to use.					
9. I felt very confident using the system.					
10. I needed to learn a lot of things before I could get going with this system.					

Table 6.1: System usability scale questionnaire

SUS Score Range	**Grade**	**Percentile Range**
84.1-100	A+	96-100
80.8-84	A	90-95
78.9-80.7	A-	85-89
77.2-78.8	B+	80-84
74.1-77.1	B	70-79
72.6-74	B-	65-69
71.1-72.5	C+	60-64
65-71	C	41-59
62.7-64.9	C-	35-40
51.7-62.6	D	15-34
0-51.7	F	0-14

Table 6.2: Curved grading scale interpretation of SUS scores according to Sauro [28]

tracking and how the implemented timeline worked. The testers said the issues were major enough that they would not use the system frequently, but would like to use the tool for certain tasks especially if the troublesome parts were addressed.

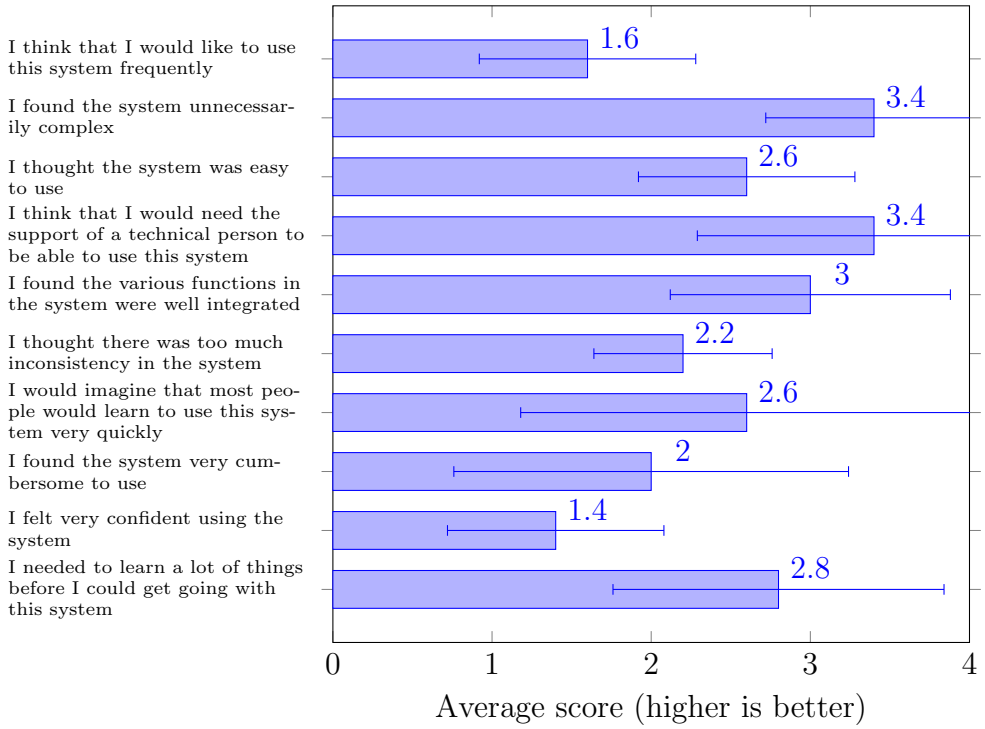


Figure 6.1: The mean SUS scores by question. Error bars represent 95% confidence interval

Better scores were attained on questions that address how easy the system is to use and how fast the testers could adapt to the system. The testers said the character posing was very easy and natural to learn and the user interface was nice and simple to get a grasp of easily. However, the testers took some time to get used to the timeline slider and how it operates, but otherwise the system is easy to learn.

When the testers were given the choice between a pinching gesture and pressing a button to grab a limb, in the end all of the testers started using the button mode. In the debriefing they said it was much faster and more precise, because the pinching mode sometimes stops pinching if the hand position becomes too awkward for the Leap Motion controller to track. Starting the pinch is also unreliable if the hands are on the edge of the Leap Motion

controller's range. Releasing the pinch also causes some residual movement in the limbs unless the user can keep his or her hand very steady while letting go. One user wanted to use the pinching much more than the button, because he felt pinching the limbs to manipulate them is more natural than having to press a button and just depend on the index finger's motions. The tester said positional manipulation was fine with the button mode, but orientation is easier with pinching. In the end the tester kept trying to use the pinching mode, but eventually switched to using the button mode. At the time of writing, the known issues for Leap Motion controller's VR mode lists pinch tracking as unreliable, so this issue is hopefully solved in the future by Leap Motion updates.

When asked *"What was good in the system?"*, one tester answered posing the character was intuitive, fun and easy once they learned how to do it. Another tester also noted how intuitive especially the pinching manipulation was, while also appreciating how the user interface displayed the basic functions. Two testers especially liked how the Oculus Rift made it feel like the character is there and it can be looked at from different viewpoints easily. The button mode to grab the character was also stated to work very well.

To the question *"What was bad in the system?"* three testers found the keyframe handling lacking. They either had trouble with the timeline or keeping track on where the character is positionally in the scene between keyframes. Hand responsiveness came up as lacking and a tester had trouble rotating the character at the hips, because the rotation comes from the palm and to rotate the palm the whole arm has to be moved. One tester had problems with placing the character's feet on the floor.

To *"What would you change or add?"* three testers gave quite similar answers by wanting more advanced rig features. For example, they would like to see limbs or joints locked in place and hand gestures directly mapped to the character's hand. A ghost that shows the character's position and pose in the previous keyframe was a wanted feature. A better timeline in general came up in the answers and one tester would like to see an indicator that tells where the Oculus Rift positional tracking camera is, so it is not blocked by hands as easily.

"If the system could work perfectly, how would you integrate it into your tool palette and workflow?" was met with answers from two opposite ends. A majority of four testers would most likely use it for rough draft work, initial key framing or testing out deformations in the rigging stage. However, one tester would have liked to try polishing existing animations made in some other software.

The question *"For what sort of animation purpose or what work stage this sort of system would work best?"* had similar answers as the previous questions. While rough work was on top, there were suggestions to use it as a previewing tool for artists, so they could observe their models with it. Also editing existing motion clips came up again, from a different tester than in the previous question.

During the debriefing the testers mentioned fatigue as a possible issue. Since the Leap Motion controller is attached to the Oculus Rift, it is not advisable to rest one's elbows on a table, because the table will interfere with the tracking quality. This means the user's hands have to be unsupported when using the system, unless the chair happens to have suitable arm rests. On the other hand, upper body movement is quite important if the user wants to fully utilize the Oculus Rift's positional tracking features, so the arm rests would not help much in the long run. Additionally, if using the pinching motion, the user's hands have to be kept in a pose that makes the finger as visible to the Leap Motion controller as possible. This hand pose can get a bit tiring after a while. Using the button mode is easier on the hand, but the arm fatigue is still a problem.

The testers also reported problems with the system's floor implementation. Sometimes the character's feet jumped between different pose configurations while the character was supposed to be stationary. The testers liked how the character takes the floor into account when solving the pose, but the floor implementation needs some work so it is more intuitive to use. As of now, the user can select between a mode that locks the part of the body that contacts the floor in place, or the body part just slides on the floor. The locking system was good when the joint rotations were being manipulated, but it made changing the floor touching body part's position difficult. To change the position, the user has to disable the floor locking mode or alternatively lift the touch part off the floor and place it elsewhere. The testers also noted that the character's heel can in some situations go through the floor.

Movable keyframes were the most wanted feature for the timeline. While the inaccurate timeline control when using the Leap Motion control is also a problem, being able to fix the keyframes later on alleviates such problems. In general the timeline was found to be a bit difficult to use and some another approach to the time manipulation problem should probably be thought of, since the task is a quite precise one and the implemented timeline is too rough for it.

A suggested way by a tester to handle viewpoint movement is to create

a turntable under the character that can be used to rotate the character in front of the user. This would not affect the animation data, but would make it possible to change the viewpoint with the Leap Motion controller, so the user does not have to switch to using the gamepad for camera control.

As a positive thing the testers said the system keeps things simple, which makes it quite easy to learn. It did not take long for the users to get proficient enough to do animations they wanted. The provided five minutes was a bit too short to get a good handle on how to best use the Leap Motion controller, but the system's interface was easy to learn in that time. The implemented features were easy to use with the provided controls and the user interface was intuitive, except for the timeline. There were no problems when the users had to use a gamepad and Leap Motion controller alternatively.

Chapter 7

Conclusions and future work

The goal was to create an animation tool that should feel natural for the user and be more productive than the dominant keyboard and mouse combination. In this chapter we ponder the results and see how well we managed to meet our goals.

7.1 Conclusions

The grade D from SUS questionnaire does not look good on paper and the reason for the grade became clear when discussing with testers about the problems they had. The tracking for hands was too unreliable for the testers to feel like they would like to use the system frequently. They did not feel confident when using the tool, because at any moment the hands could disappear or jerk in some direction and mess up the task at hand, especially when using the timeline. Using the timeline with the Xbox 360 controller is easier and more precise than with the Leap Motion, because the hands do shake a bit and the timeline work is quite precise. Indeed, the slider probably was not ideal for this sort of timeline work and there should be another approach on how to solve the problem. One idea is presented in the next section 7.2.

On the other hand, the testers said the posing with Leap Motion felt very natural after they learned the limitations of the device and did not try to do too long reaching motions. They said they could see use for such tools in tasks such as rough initial keyframing, manipulating existing animation data or using it as a tool for artists to inspect their model in 3D. As long as some needed features were implemented and the hand tracking was more reliable, they could integrate it into their tool palette.

The hand tracking is hopefully improving over time, since the Leap Motion VR mode is still in beta and the tracking quality is not as good as with the original desktop mode. We will look into its development with interest.

The head tracking worked very well in conjunction with object manipulation by hand tracking. While the user had two hands grabbing the character, he still could change the viewpoint without having to free one hand to use the gamepad. The users expressed how natural it feels to work with an virtual object that looks like it is really there.

In the end, the tool performed much better in the posing tasks than more application control -like tasks such as the keyframe handling and timeline scrubbing. Combining the tool with the keyboard and mouse using the image passthrough feature as an aide is something that could increase the tool's usability and productivity by a large margin. The technology is not quite there yet because of the reliability issues, but the issues are likely to be alleviated by software updates.

7.2 Future work

Although the system performed fairly well for posing tasks, with the present features it is not very useful, because the animation data can not be exported and the character model can only be changed in the Unity editor and the whole system needs to be built again. Testers also had some features in mind that would make the system more desirable. To make the system more useful, here are some ideas for future work:

Better timeline
: The timeline presented in this version was found to be difficult to use and too inaccurate when using the Leap Motion. Possible alternative is to implement a more traditional timeline found in current leading animation software and control it with a mouse. Since the Oculus Rift and Leap Motion combination has the passthrough ability, finding the mouse on a table would not be a problem. The computer mouse is very accurate and established for timeline control after all, so this approach should be considered.

Controlling the character's hands with hand gestures
: One obvious use for the Leap Motion controller is to control the character's hands directly. The Leap Motion SDK gives accurate joint

positions, so the character's hand's joints could be mapped to them, even if the hand was not completely human-like.

Turntable below the character

This idea came from the testers. A spinnable disc under the character that can be grabbed by Leap Motion hands would make rotating around the character faster and more intuitive than using the Xbox 360 controller.

Animation data import and export

To make the system truly useful, the resulting animation needs to be usable elsewhere. FBX export seems like the most sensible solution. Animation data importing is also useful, so the system can be used to inspect and modify animation data made with the system before or with other software.

Model import

Right now it is not possible to import a character inside the system. The user has to use the Unity editor to import the character and build the system again. On top of that, the user has to run the ragdoll wizard, drag the relevant transforms in place and tell the PoseableCharacter script which transforms are to be used as IK chain starting points.

Locking joints and body parts in place

The testers would have liked to lock certain body parts in place, much like the floor does with the limbs that come in contact with it. A button press that locks the grabbed limb in place should be implemented.

Bibliography

[1] Balakrishnan, R., Baudel, T., Kurtenbach, G., and Fitzmaurice, G. The Rockin'Mouse: integral 3D manipulation on a plane. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems* (1997), ACM, pp. 311–318.

[2] Bell, G. Bell's trackball. *Written as part of the "flip" demo to demonstrate the Silicon Graphics (now SGI) hardware* (1988).

[3] Bier, E. A. Skitters and jacks: Interactive 3D positioning tools. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics* (New York, NY, USA, 1987), I3D '86, ACM, pp. 183–196.

[4] Boritz, J., and Booth, K. S. A study of interactive 3D point location in a computer simulated virtual environment. In *Proceedings of the ACM symposium on Virtual reality software and technology* (1997), ACM, pp. 181–187.

[5] Boritz, J., and Booth, K. S. A study of interactive 6 DOF docking in a computerised virtual environment. In *Virtual Reality Annual International Symposium, 1998. Proceedings., IEEE 1998* (1998), IEEE, pp. 139–146.

[6] Brooke, J. SUS - a quick and dirty usability scale. *Usability evaluation in industry 189*, 194 (1996), 4–7.

[7] Chen, J., Izadi, S., and Fitzgibbon, A. Kinêtre: animating the world with the human body. In *Proceedings of the 25th annual ACM symposium on User interface software and technology* (2012), ACM, pp. 435–444.

[8] Chen, M., Mountford, S. J., and Sellen, A. A study in interactive 3-D rotation using 2-D control devices. *SIGGRAPH Comput. Graph. 22*, 4 (June 1988), 121–129.

[9] Colgan, A. How does the Leap Motion controller work?, Aug. 2014. `http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/`. Accessed: 23 January 2015.

[10] Feng, T.-C., Gunawardane, P., Davis, J., and Jiang, B. Motion capture data retrieval using an artist's doll. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on* (2008), IEEE, pp. 1–4.

[11] Fröhlich, B., and Plate, J. The cubic mouse: A new device for three-dimensional input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2000), CHI '00, ACM, pp. 526–531.

[12] Gallo, L. A glove-based interface for 3D medical image visualization. In *Intelligent interactive multimedia systems and services*. Springer, 2010, pp. 221–230.

[13] Hand, C. A survey of 3-D input devices. *DMU CS TR94/2* (1994).

[14] Hand, C. A survey of 3D interaction techniques. *Computer Graphics Forum 16*, 5 (1997), 269–281.

[15] Hutchins, E. L., Hollan, J. D., and Norman, D. A. Direct manipulation interfaces. *Human–Computer Interaction 1*, 4 (1985), 311–338.

[16] Ishigaki, S., White, T., Zordan, V. B., and Liu, C. K. Performance-based control interface for character animation. In *ACM Transactions on Graphics (TOG)* (2009), vol. 28, ACM, p. 61.

[17] Ishii, H., and Ullmer, B. Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1997), CHI '97, ACM, pp. 234–241.

[18] Jacob, R. J. K., Sibert, L. E., McFarlane, D. C., and Mullen, Jr., M. P. Integrality and separability of input devices. *ACM Trans. Comput.-Hum. Interact. 1*, 1 (Mar. 1994), 3–26.

[19] Jacobson, A., Panozzo, D., Glauser, O., Pradalier, C., Hilleges, O., and Sorkine-Horning, O. Tangible and modular input device for character articulation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH) 33*, 4 (2014).

[20] Jankowski, J., and Hachet, M. A Survey of Interaction Techniques for Interactive 3D Environments. In *Eurographics 2013 - STAR* (Girona, Spain, May 2013).

[21] Jones, P. E. Three-dimensional input device with six degrees of freedom. *Mechatronics 9*, 7 (1999), 717 – 729.

[22] Margono, S., and Shneiderman, B. A study of file manipulation by novices using commands vs, direct manipulation. *Sparks of Innovation in Human-computer Interaction* (1993), 39.

[23] Mixamo Inc. Mixamo Alpha-model, Mar. 2015. `https://www.mixamo.com/editor/new/729?character_id=110749`. Accessed: 1 April 2015.

[24] Moeslund, T. B., Hilton, A., and Krüger, V. A survey of advances in vision-based human motion capture and analysis. *Computer vision and image understanding 104*, 2 (2006), 90–126.

[25] Oh, J.-Y., and Stuerzlinger, W. Moving objects with 2D input devices in CAD systems and desktop virtual environments. In *Proceedings of Graphics Interface 2005* (2005), Canadian Human-Computer Communications Society, pp. 195–202.

[26] Poupyrev, I., Ichikawa, T., Weghorst, S., and Billinghurst, M. Egocentric object manipulation in virtual environments: empirical evaluation of interaction techniques. In *Computer Graphics Forum* (1998), vol. 17, Wiley Online Library, pp. 41–52.

[27] Rolland, J. P., Holloway, R. L., and Fuchs, H. Comparison of optical and video see-through, head-mounted displays. In *Photonics for Industrial Applications* (1995), International Society for Optics and Photonics, pp. 293–307.

[28] Sauro, J., and Lewis, J. R. *Quantifying the user experience: Practical statistics for user research*. Elsevier, 2012.

[29] Shneiderman, B., and Plaisant, C. *Designing the User Interface: Strategies for Effective Human-Computer Interaction (5th Edition)*. Pearson Addison Wesley, 2010.

[30] Shoemake, K. ARCBALL: A user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of the Conference on Graphics Interface '92* (San Francisco, CA, USA, 1992), Morgan Kaufmann Publishers Inc., pp. 151–156.

[31] Smith, G., Stuerzlinger, W., Salzman, T., Watson, B., and Buchanan, J. 3D scene manipulation with 2D devices and constraints. In *Graphics Interface* (2001), vol. 1, pp. 135–142.

[32] Sutherland, I. E. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I* (1968), ACM, pp. 757–764.

[33] Taylor, P. Non-iterative, closed-form, inverse kinematic chain solver (NCF IK). In *Game Programming Gems 8*, A. Lake, Ed. Cengage Learning, 2010, pp. 141–151.

[34] Unity Technologies. Unity documentation: Positioning GameObjects, Mar. 2015. `http://docs.unity3d.com/Manual/PositioningGameObjects.html`. Accessed: 17 March 2015.

[35] Watt, A., and Watt, M. Advanced animation and rendering techniques. *Addison-Wesley* (1992).

[36] Welman, C. Inverse kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Fraser University, 1993.

[37] Yoshizaki, W., Sugiura, Y., Chiou, A. C., Hashimoto, S., Inami, M., Igarashi, T., Akazawa, Y., Kawachi, K., Kagami, S., and Mochimaru, M. An actuated physical puppet as an input device for controlling a digital manikin. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2011), ACM, pp. 637–646.

[38] Zhai, S. User performance in relation to 3D input device design. *SIGGRAPH Comput. Graph. 32*, 4 (Nov. 1998), 50–54.

[39] Zimmerman, T. G., Lanier, J., Blanchard, C., Bryson, S., and Harvill, Y. A hand gesture interface device. In *Proceedings of the SIGCHI/GI Conference on Human Factors in Computing Systems and Graphics Interface* (New York, NY, USA, 1987), CHI '87, ACM, pp. 189–192.